DNN Latency Sequencing: Extracting DNN Architectures from Intel SGX Enclaves with Single-Stepping Attacks

Minkyung Park[1], Zelun Kong[1], Dave (Jing) Tian[2], Z. Berkay Celik[2], Chung Hwan Kim[1]

[1]University of Texas at Dallas
[2]Purdue University

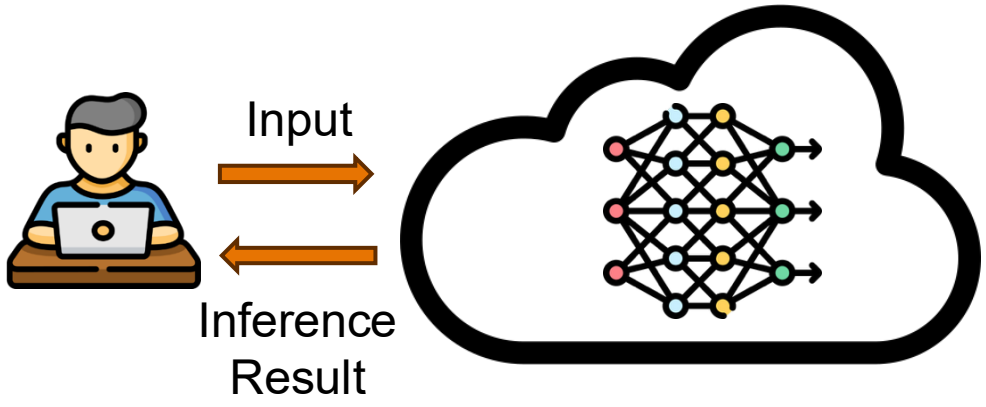
Deep Neural Networks (DNNs)

- Widely used across many systems
  - e.g., image recognition, audio analysis, natural language processing

Deep Neural Networks (DNNs)

- Widely used across many systems
  - e.g., image recognition, audio analysis, natural language processing

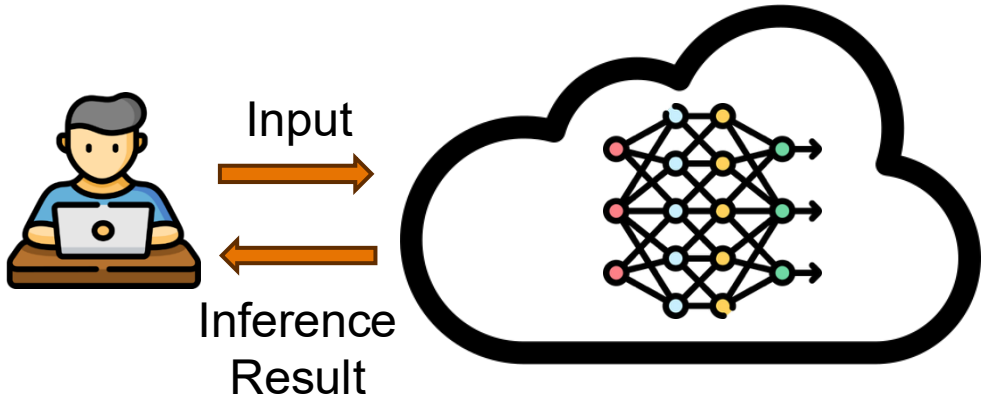
- Machine Learning as a Service (MLaaS)



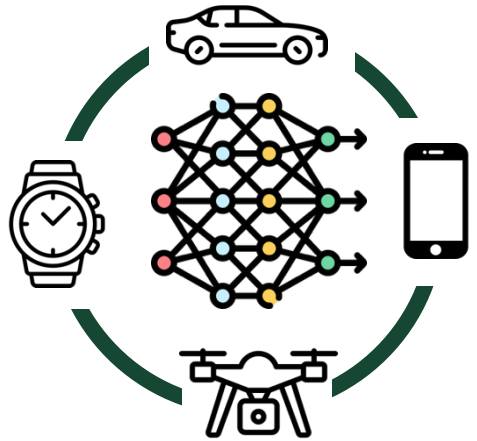
Deep Neural Networks (DNNs)

- Widely used across many systems
  - e.g., image recognition, audio analysis, natural language processing

- Machine Learning as a Service (MLaaS)



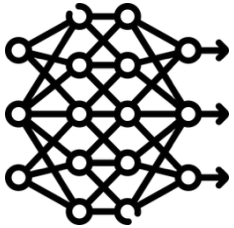
- On Device Deep Learning



DNN Models

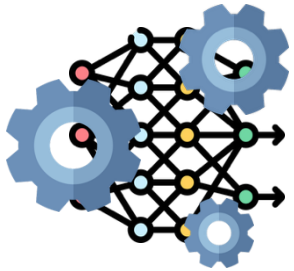
- System performance is largely determined by the DNN model

Structure



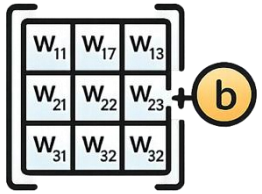
- #Layer
- Layer type

Hyperparameter



- Kernel size
- Pool size

Parameter



DNN Models

- Serve as valuable intellectual property (IP)
- Designing and training a model is computation-intensive and time-consuming

Input Acquisition & Labeling

GPU, Power, Time Cost

Domain Expertise

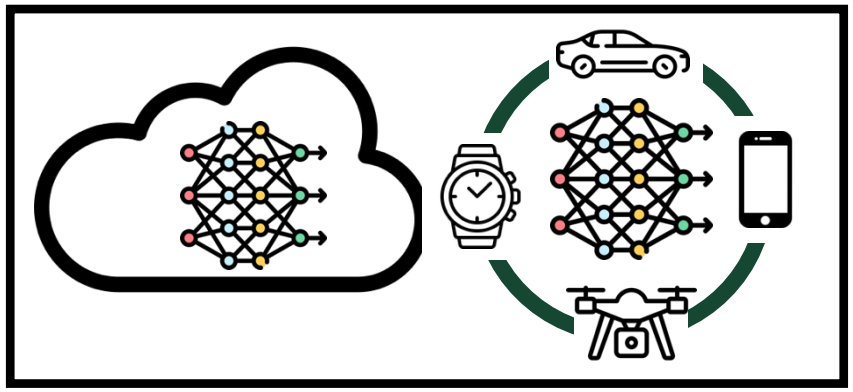
Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - Architecture stealing
  - Parameter stealing

Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - Architecture stealing
  - Parameter stealing

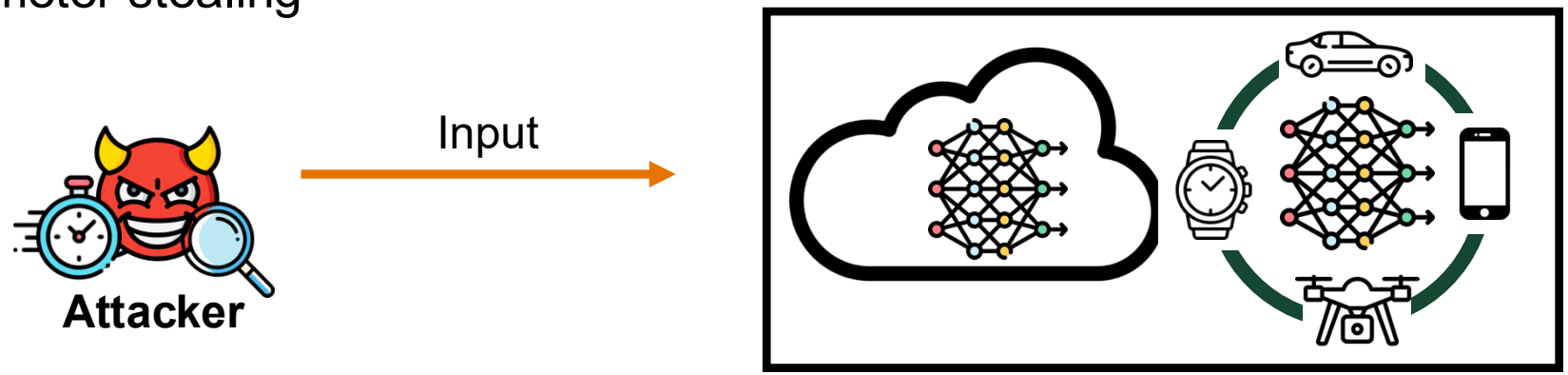
Attacker



Victim Models in Black Box

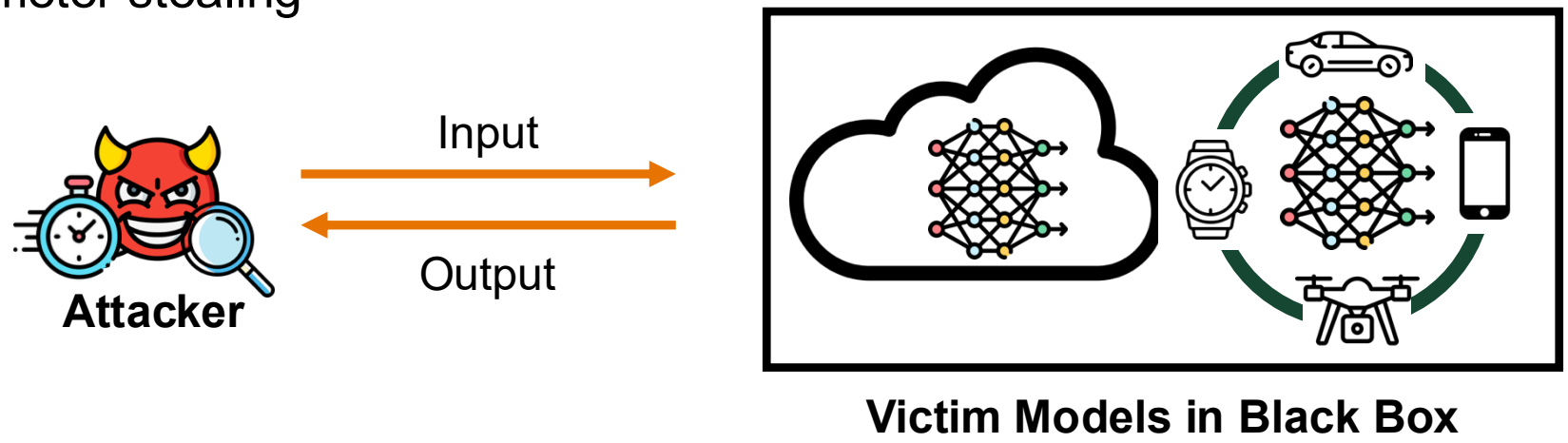
Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - Architecture stealing
  - Parameter stealing



Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - Architecture stealing
  - Parameter stealing



Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - **Architecture stealing**
  - Parameter stealing

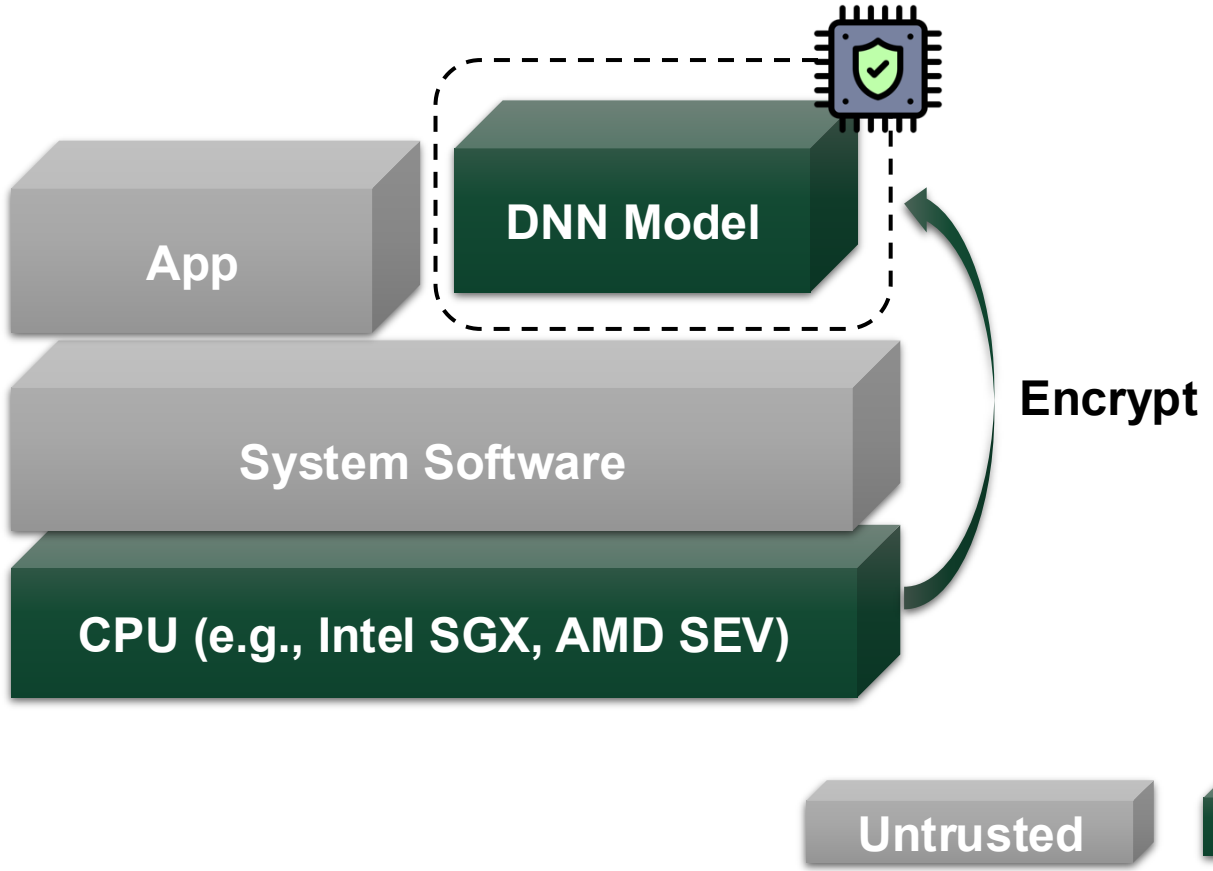
Model Extraction Attack: Architecture Stealing

- A type of attack to steal DNN architectures
  - **Architecture stealing**
  - Parameter stealing

- Why it matters
  - Enables reconstruction of a similarly performing model
  - Acts as a stepping stone for follow-on attacks

Confidential AI

- Trusted Execution Environment (TEE) protects DNN models by during its processing

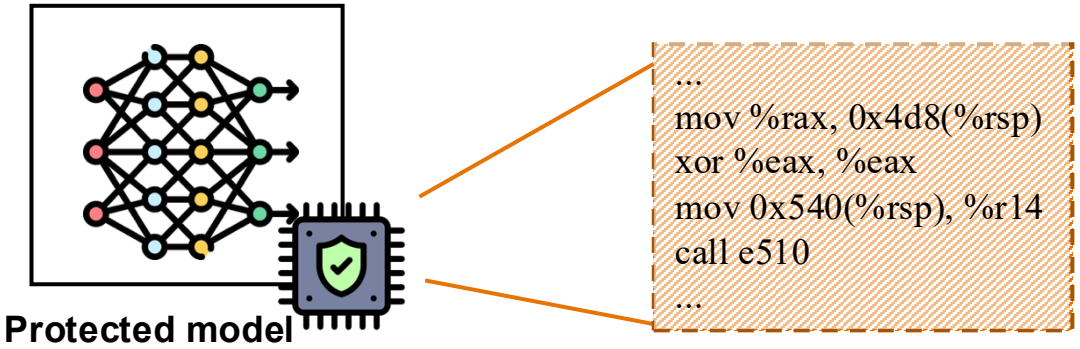


Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts

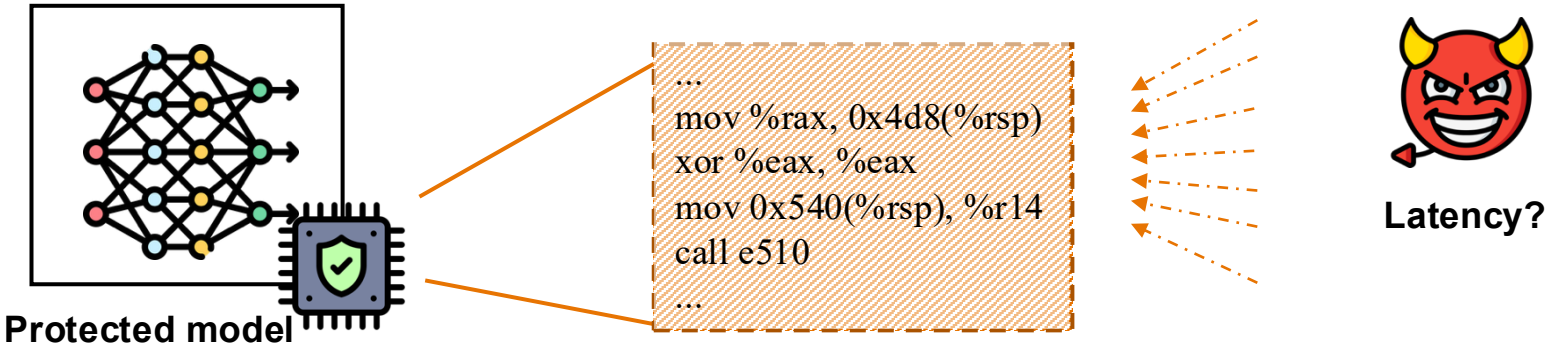
Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts



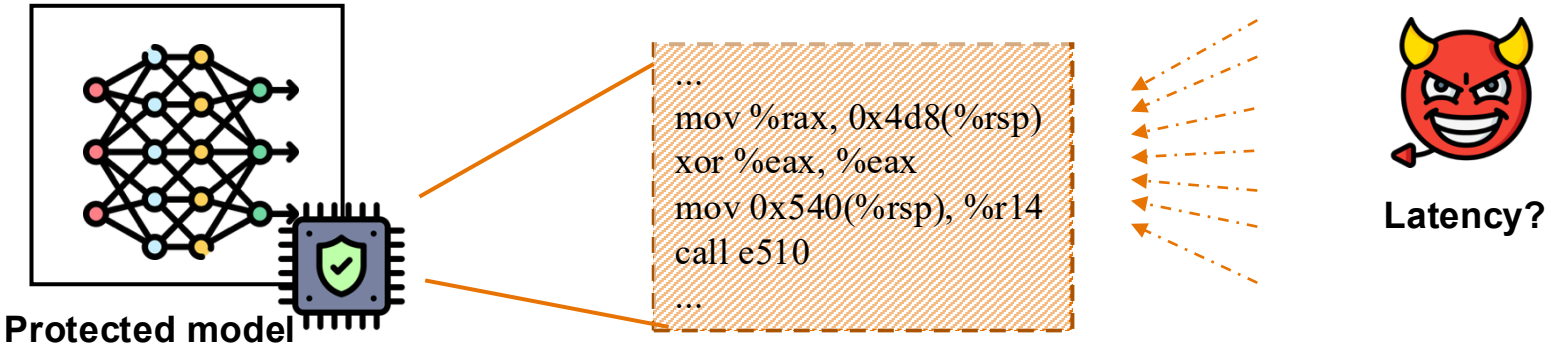
Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts



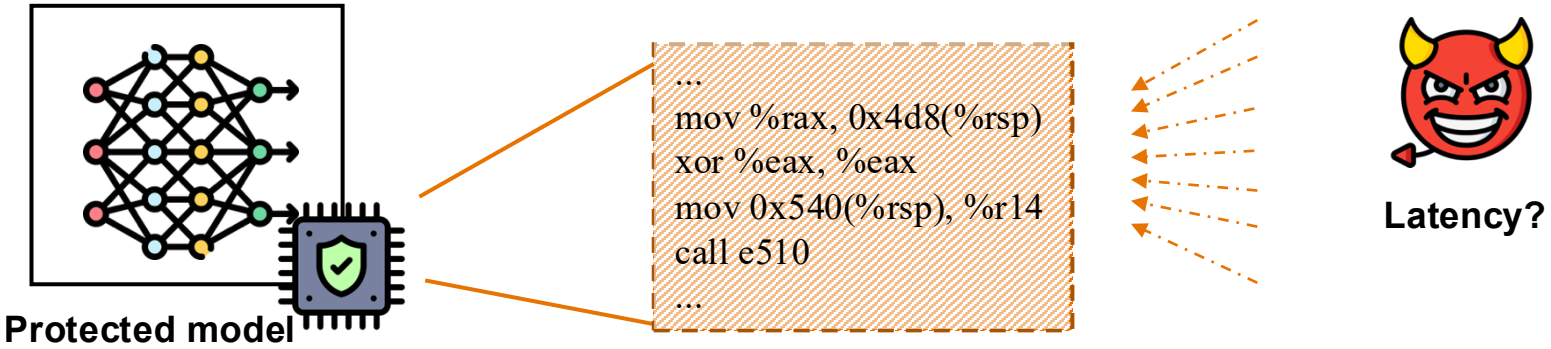
Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts



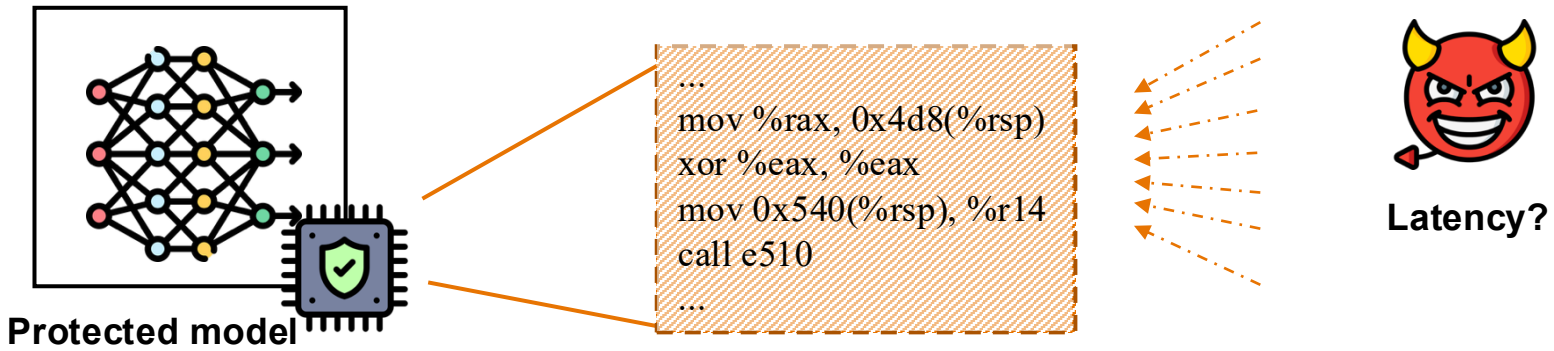
Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts



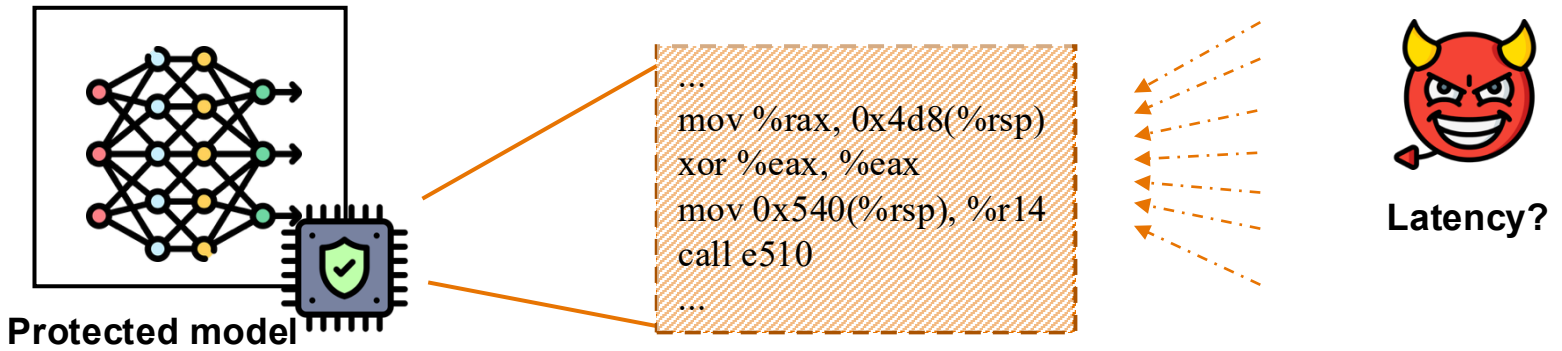
Single-Stepping Attacks on TEE

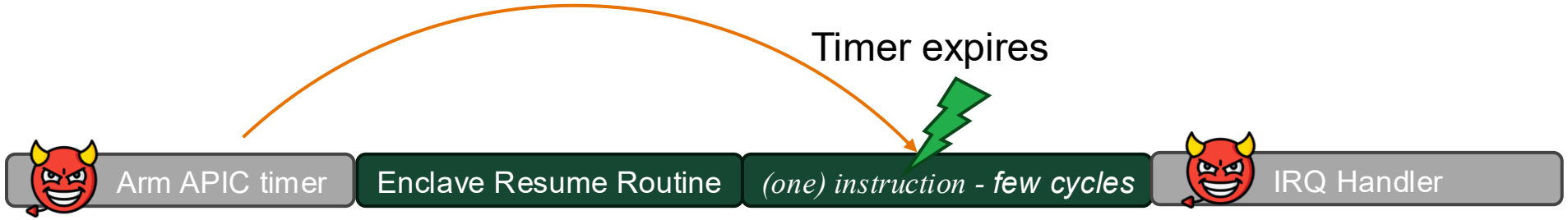
- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts



Single-Stepping Attacks on TEE

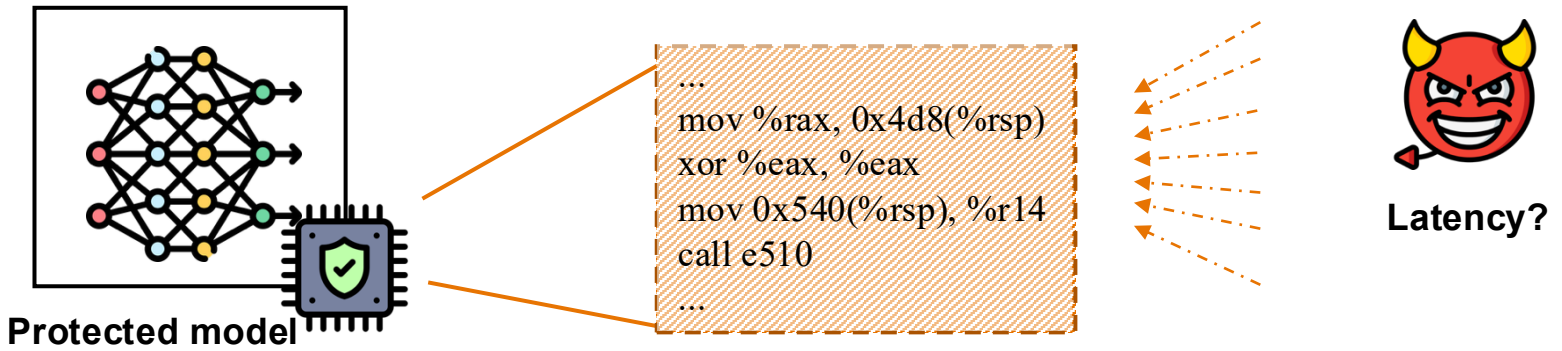
- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts

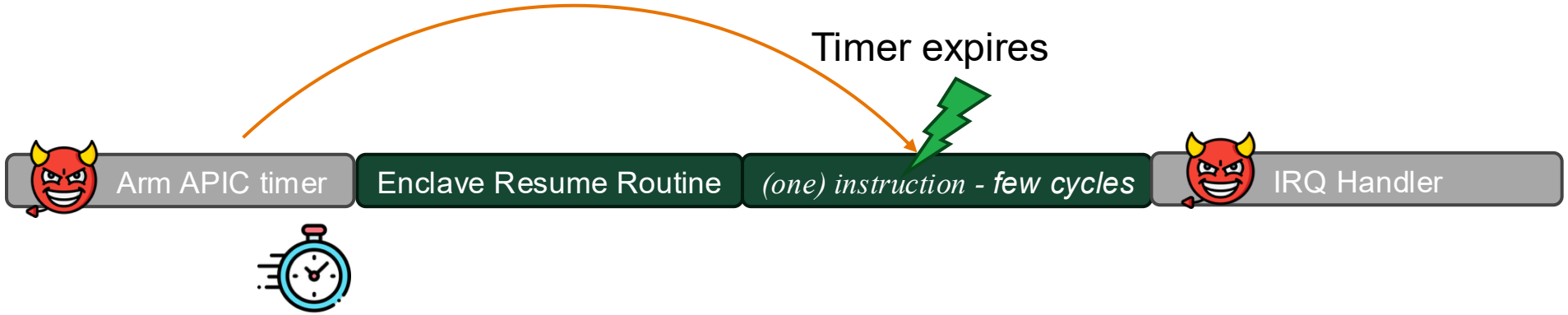




Single-Stepping Attacks on TEE

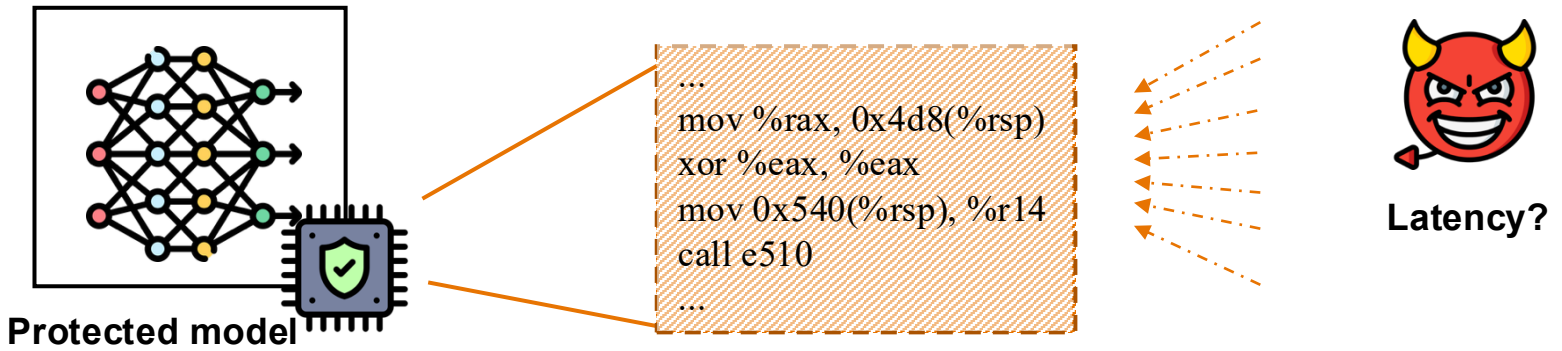
- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts

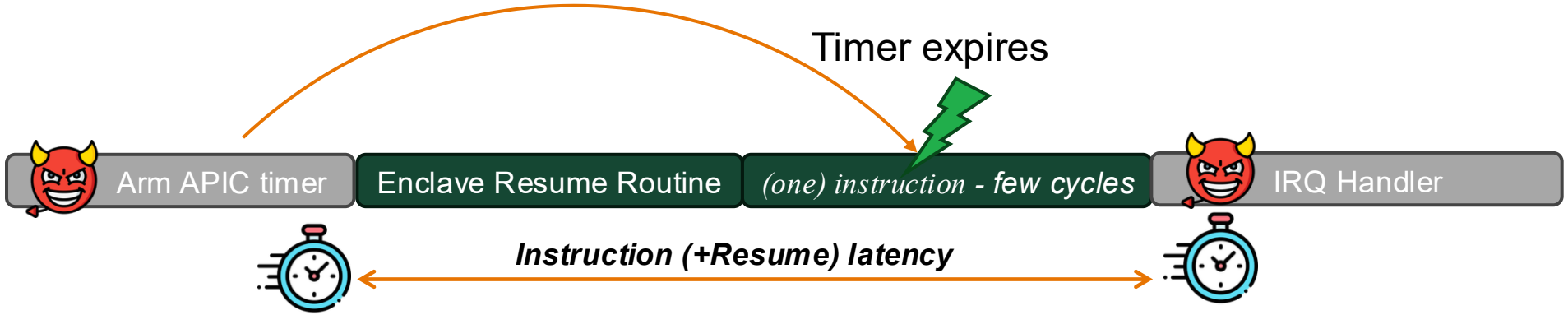




Single-Stepping Attacks on TEE

- TEE introduces new types of side-channels such as ***instruction latency***
  - Exploited using high-resolution APIC timer interrupts





Proposal: DLS (DNN Latency Sequencing)

- DLS recovers the DNN architecture protected by Intel SGX

Proposal: DLS (DNN Latency Sequencing)

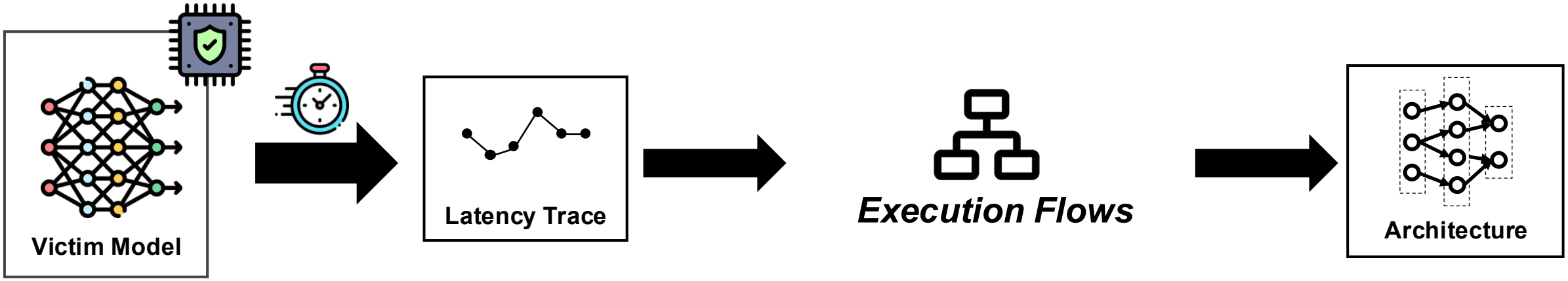
- DLS recovers the DNN architecture protected by Intel SGX
- Threat model: privileged attacker
  - Collects latency traces
  - Triggers inference process
  - Knows the used DNN library/framework

Proposal: DLS (DNN Latency Sequencing)

- DLS recovers the DNN architecture protected by Intel SGX
- Threat model: privileged attacker
  - Collects latency traces
  - Triggers inference process
  - Knows the used DNN library/framework

Proposal: DLS (DNN Latency Sequencing)

- DLS recovers the DNN architecture protected by Intel SGX
- Threat model: privileged attacker
  - Collects latency traces
  - Triggers inference process
  - Knows the used DNN library/framework



Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

Code example in Darknet (1)

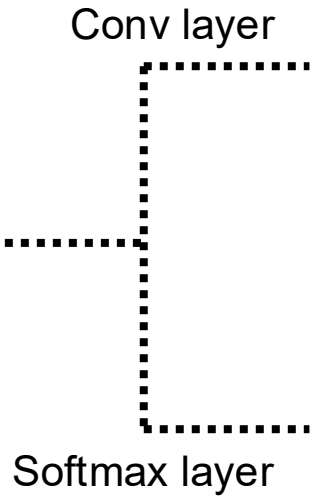
- Different layer type calls distinct sequence of functions

```
void forward_network(network *netp)
{
  for(i = 0; i < net.n; ++i){
    layer l = net.layers[i];
    l.forward(l, net);
  }
}
```

Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

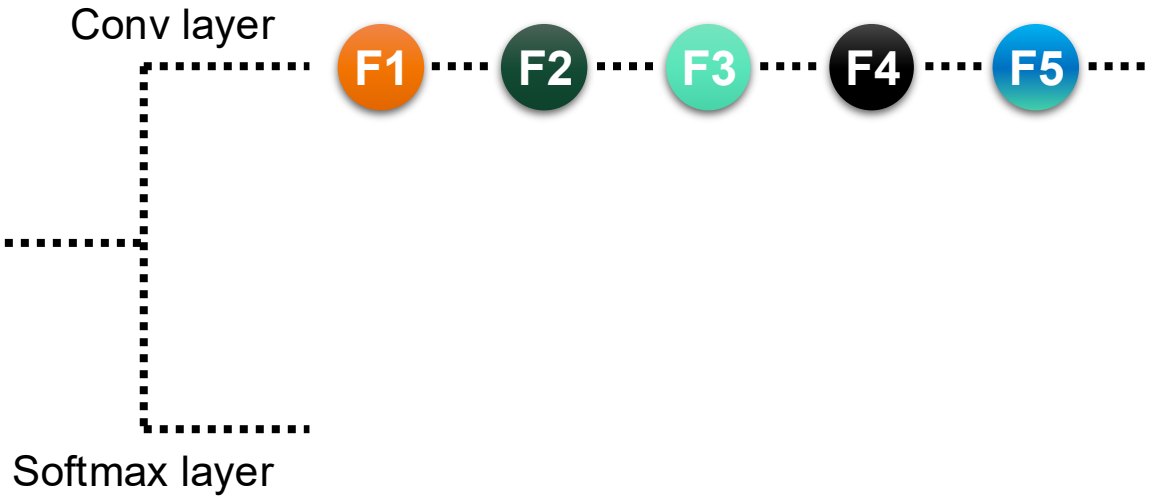
```
void forward_network(network *netp)
{
  for(i = 0; i < net.n; ++i){
    layer l = net.layers[i];
    l.forward(l, net);
  }
}
```



Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

```
void forward_network(network *netp)
{
  for(i = 0; i < net.n; ++i){
    layer l = net.layers[i];
    l.forward(l, net);
  }
}
```

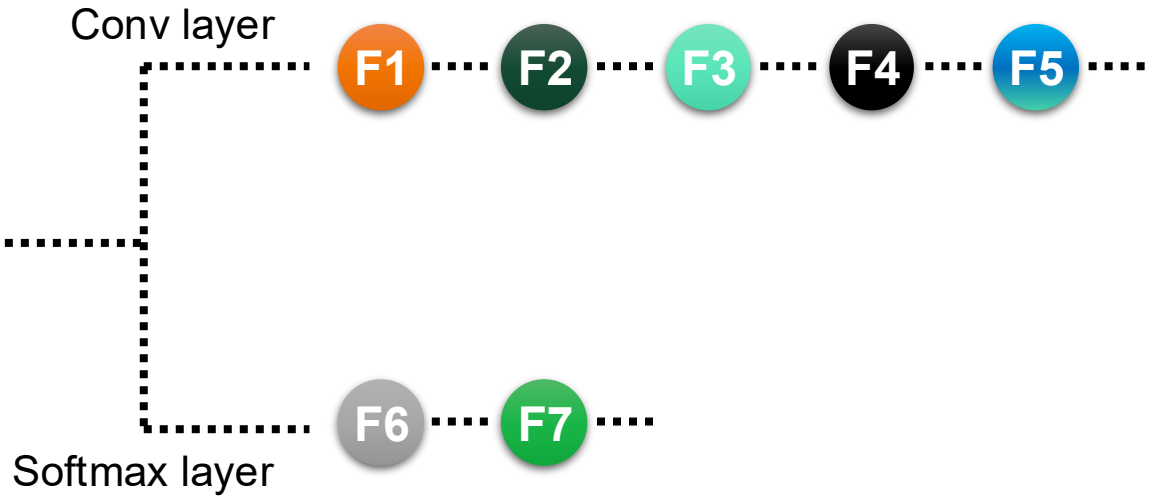


F1: `forward_convolutional_layer()`
F2: `fill_cpu()`
…
F6: `forward_software_max()`
F7: `softmax()`

Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

```
void forward_network(network *netp)
{
  for(i = 0; i < net.n; ++i){
    layer l = net.layers[i];
    l.forward(l, net);
  }
}
```

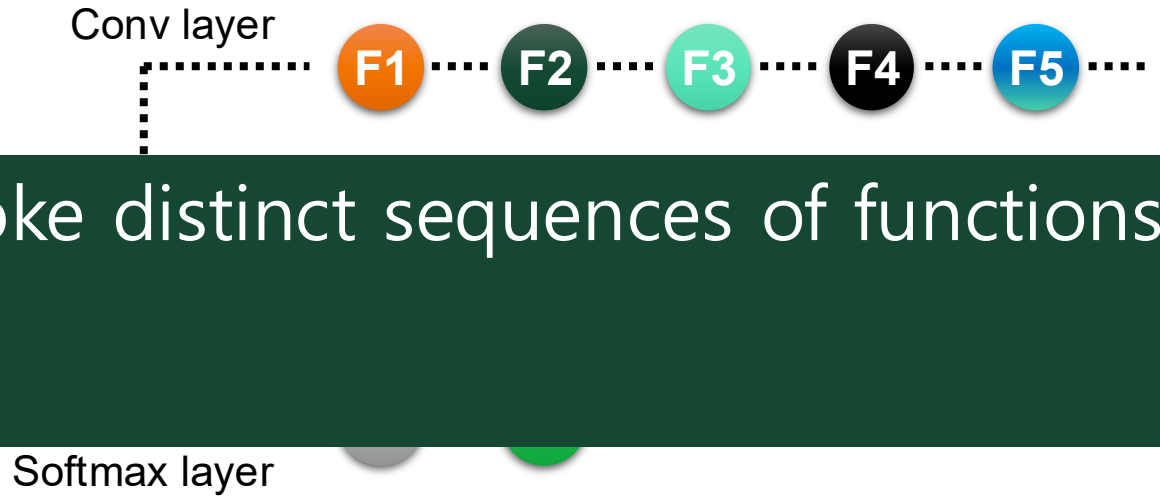


F1: `forward_convolutional_layer()`
F2: `fill_cpu()`
…
F6: `forward_software_max()`
F7: `softmax()`

Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

```
void forward_network(network *netp)
{
    for(i = 0; i < net.n; ++i){
```



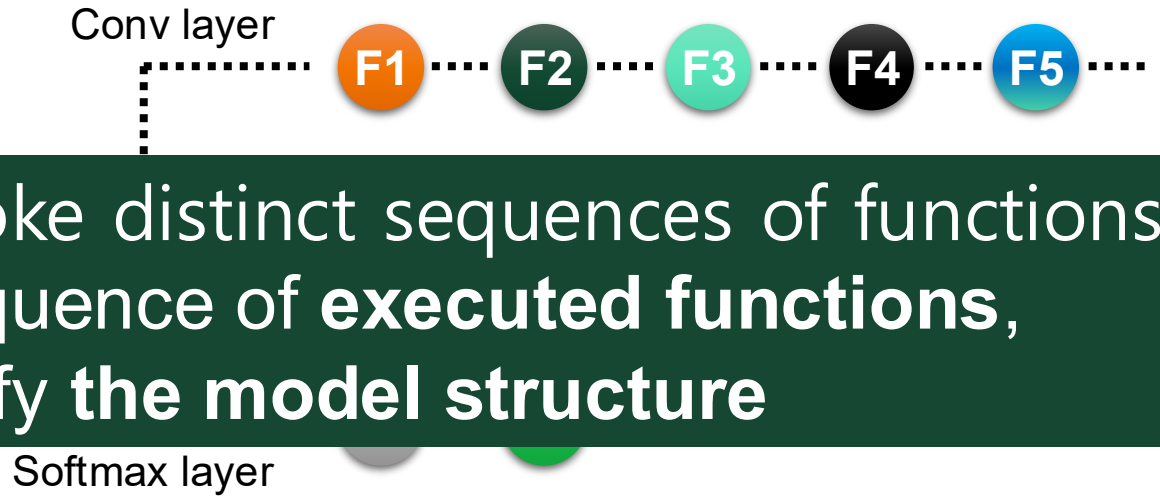
Different layer types invoke distinct sequences of functions

F1: `forward_convolutional_layer()`
F2: `fill_cpu()`
…
F6: `forward_software_max()`
F7: `softmax()`

Code example in Darknet (1)

- Different layer type calls distinct sequence of functions

```
void forward_network(network *netp)
{
    for(i = 0; i < net.n; ++i){
```



Different layer types invoke distinct sequences of functions
➡ By identifying the sequence of **executed functions**,
we can identify **the model structure**

F1: `forward_convolutional_layer()`
F2: `fill_cpu()`
…
F6: `forward_software_max()`
F7: `softmax()`

Code example in Tensorflow Lite (2)

- Certain hyperparameters influence a control flow
  - e.g., the number of loop iterations or a branch taken

```
inline void Conv(...)
{
  for (int i = 0; i < filter_height; ++i)
    for (int j = 0; j < filter_width; ++j)
      for (int k = 0; k < input_depth; ++k) {
         float iv = input_data[Offset(...)];
         float fv = filter_data[Offset(...)];
         total += (input_value * filter_value);
      }
   }
}
```

Code example in Tensorflow Lite (2)

- Certain hyperparameters influence a control flow
  - e.g., the number of loop iterations or a branch taken

```
inline void Conv(...)
{
  for (int i = 0; i < filter_height; ++i)
    }
}
```

Depending on the value of hyperparameter, the control flow changes

Code example in Tensorflow Lite (2)

- Certain hyperparameters influence a control flow
  - e.g., the number of loop iterations or a branch taken

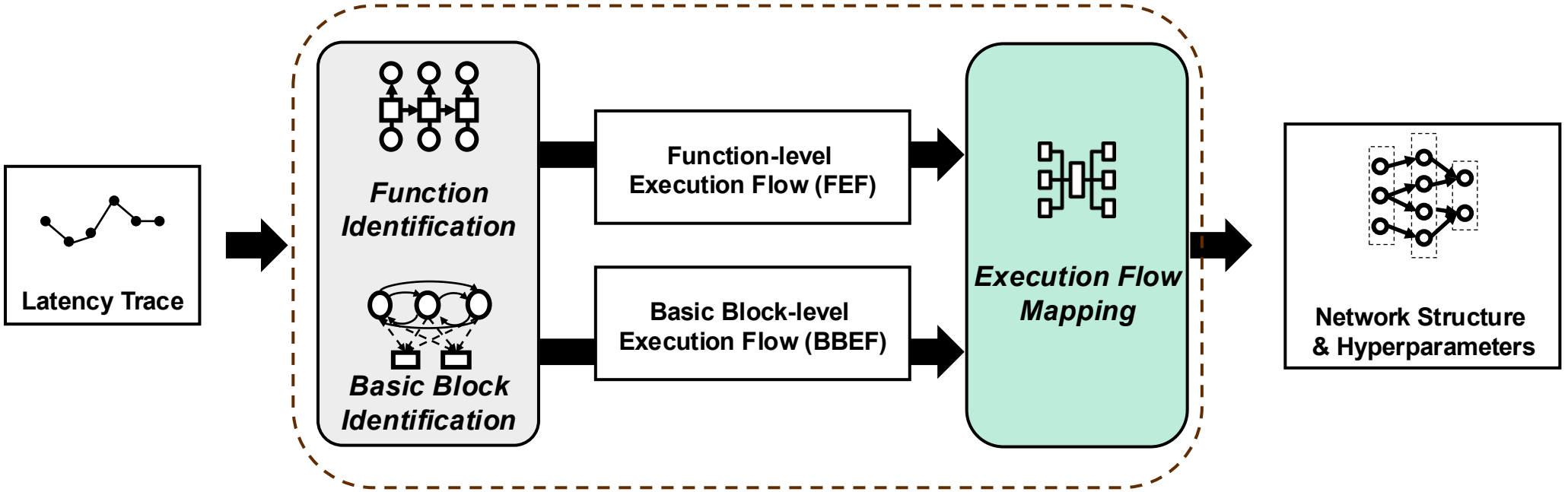
```
inline void Conv(...)
{
  for (int i = 0; i < filter_height; ++i)
    }
}
```

Depending on the value of hyperparameter, the control flow changes

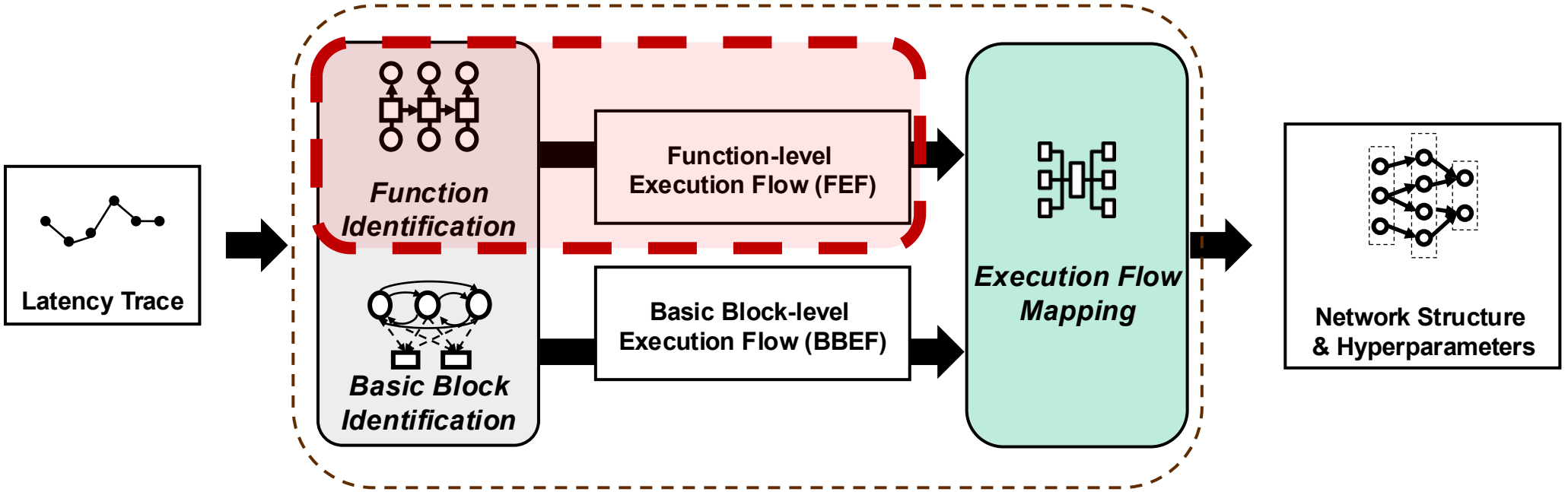
➡ By identifying the sequence of **executed *basic blocks**,
we can identify **the hyperparameters**

*Basic Block? A sequence of instructions w/
- a single entry at the beginning and
- a single exit at the end

Execution Flow Identification

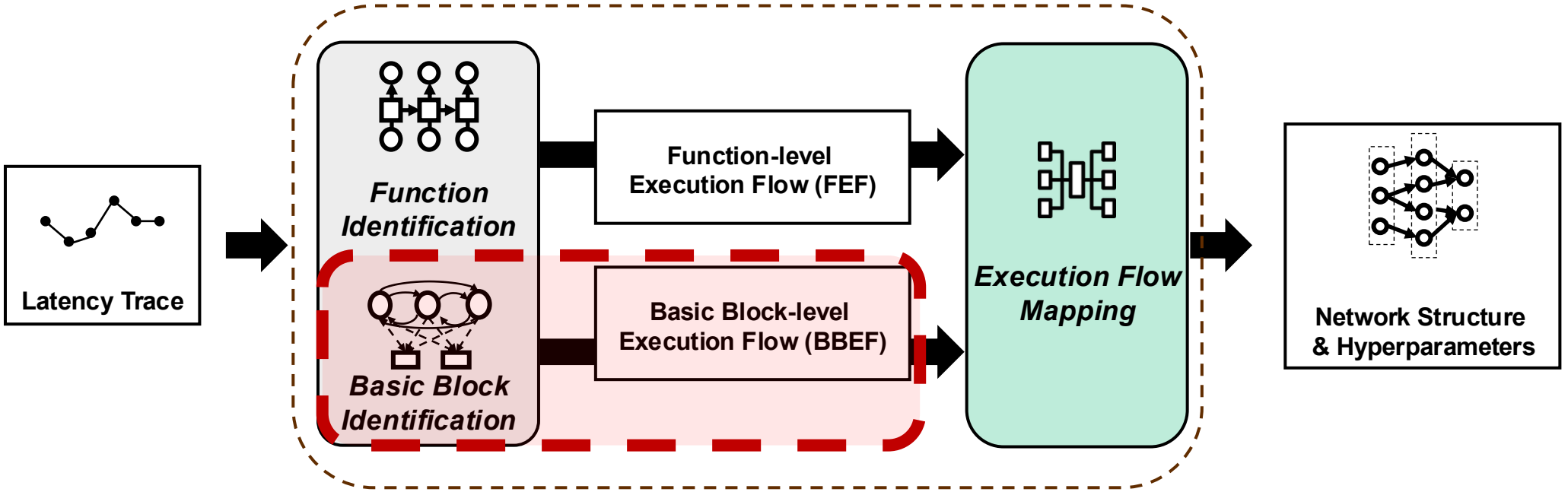


Execution Flow Identification



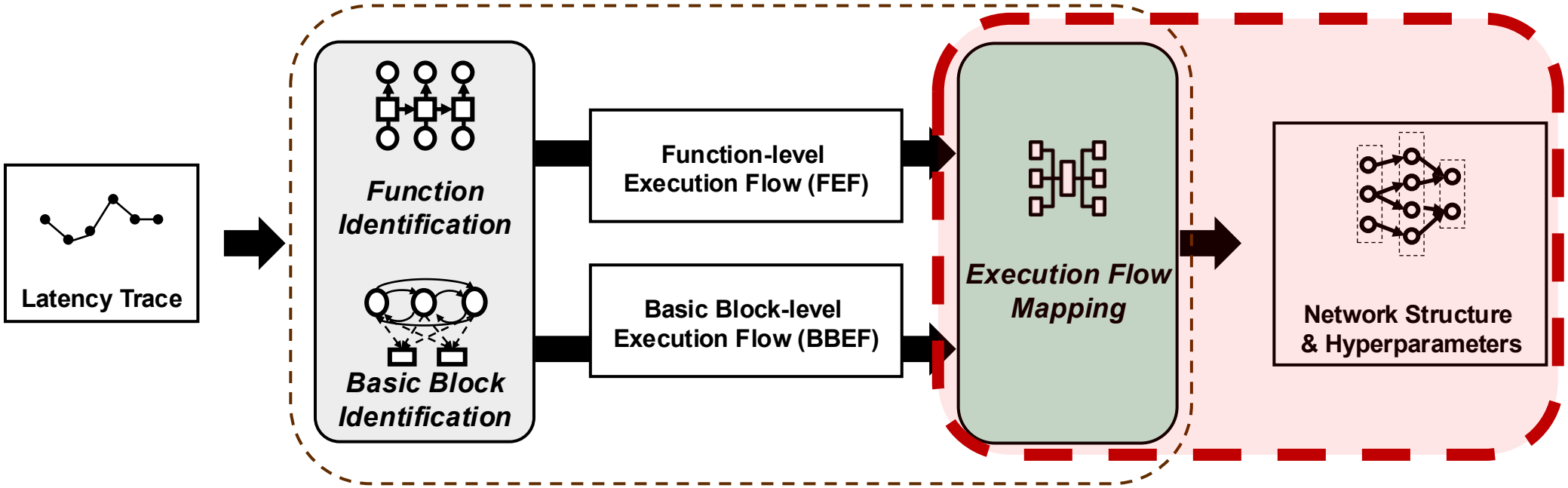
- Identifying function-level execution flows (FEFs)
  - **CNN-BiLSTM** extracts patterns and learns sequential dependencies

Execution Flow Identification



- Identifying function-level execution flows (FEFs)
  - **CNN-BiLSTM** extracts patterns and learns sequential dependencies
- Identifying basic block-level execution flows (BBEFs)
  - **Semi Hidden Markov Model (SHMM)** represents control flows as a probabilistic model

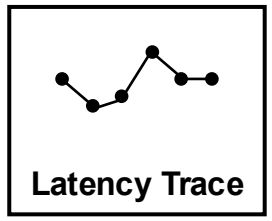
Execution Flow Identification



- Identifying function-level execution flows (FEFs)
  - **CNN-BiLSTM** extracts patterns and learns sequential dependencies
- Identifying basic block-level execution flows (BBEFs)
  - **Semi Hidden Markov Model (SHMM)** represents control flows as a probabilistic model
- Mapping the FEFs and BBEFs to architecture details

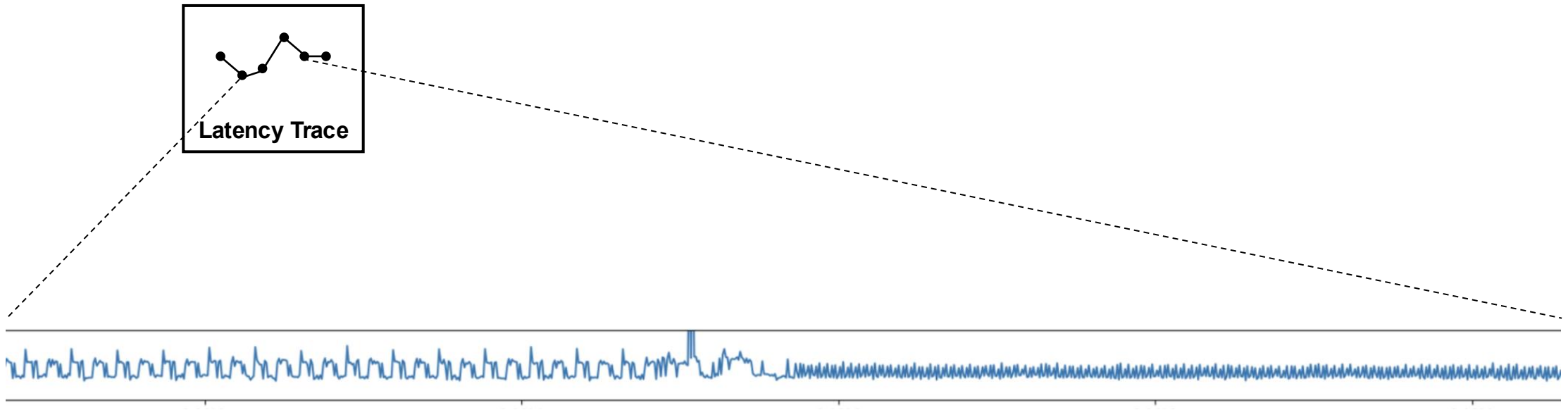
Identifying FEF

- **CNN-BiLSTM** extracts patterns and learns sequential dependencies



Identifying FEF

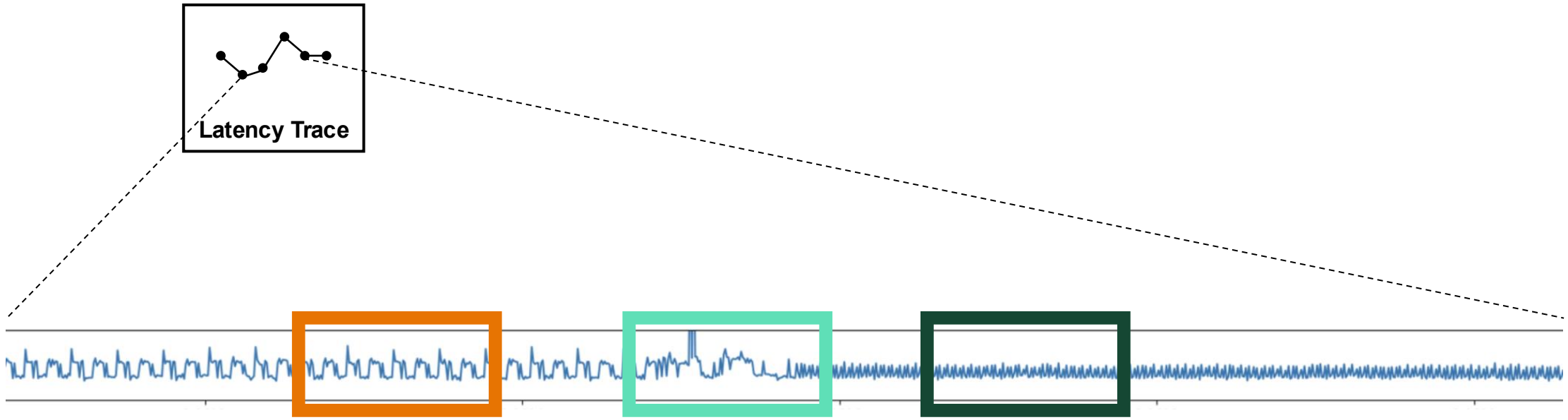
- **CNN-BiLSTM** extracts patterns and learns sequential dependencies



[A segment of latency trace]

Identifying FEF

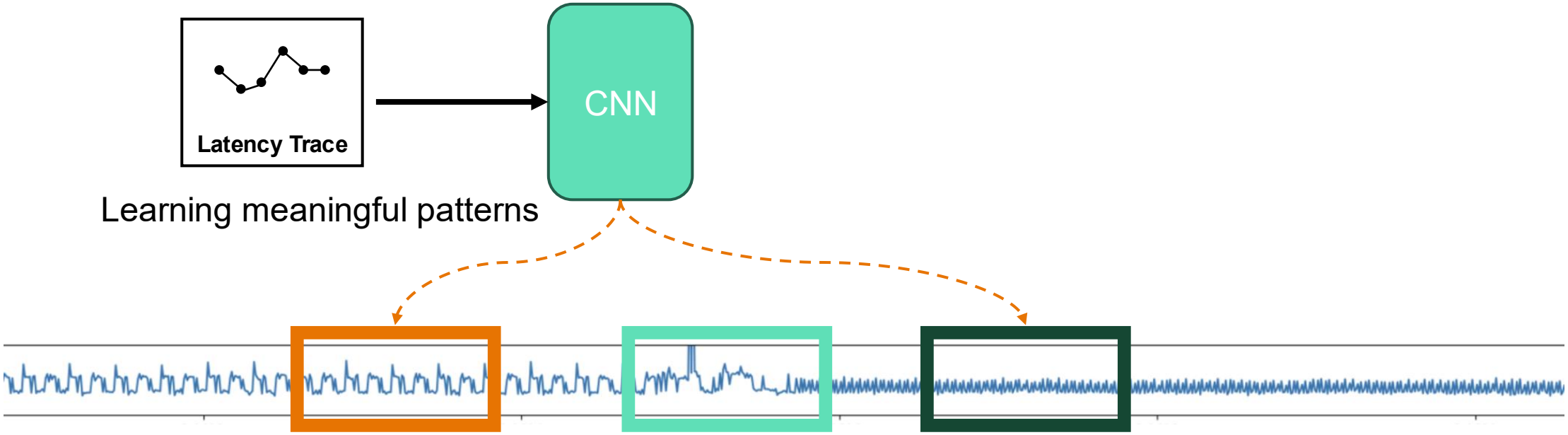
- **CNN-BiLSTM** extracts patterns and learns sequential dependencies



[A segment of latency trace]

Identifying FEF

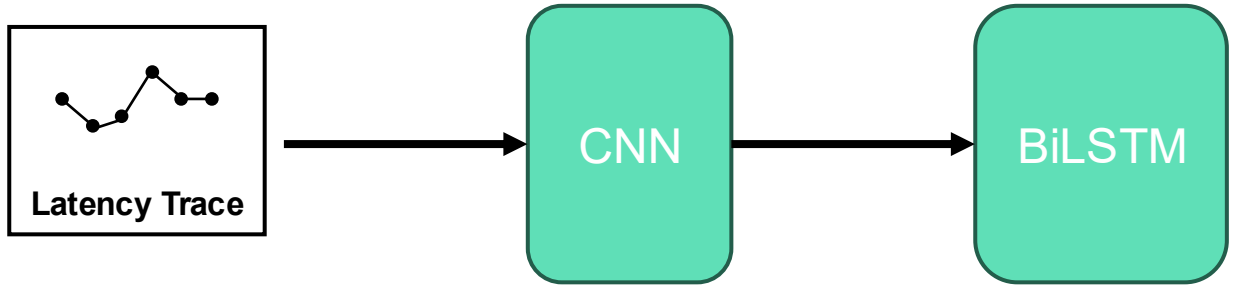
- **CNN-BiLSTM** extracts patterns and learns sequential dependencies



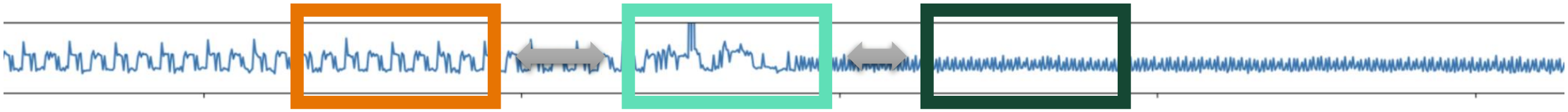
[A segment of latency trace]

Identifying FEF

- **CNN-BiLSTM** extracts patterns and learns sequential dependencies



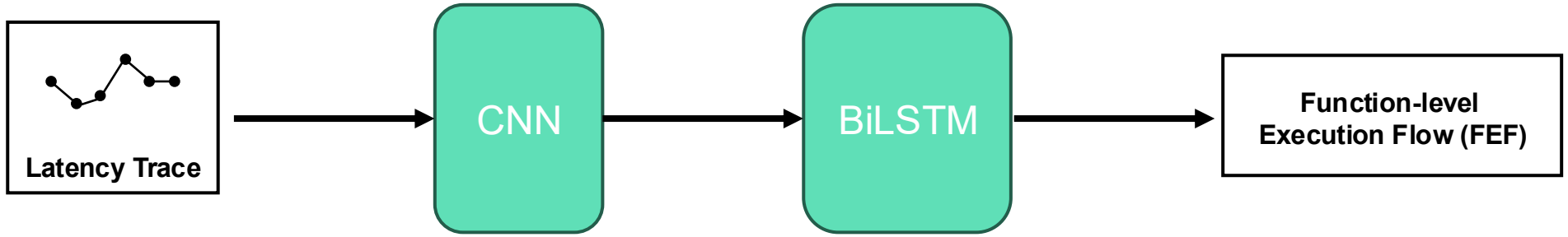
Learning sequential relationship

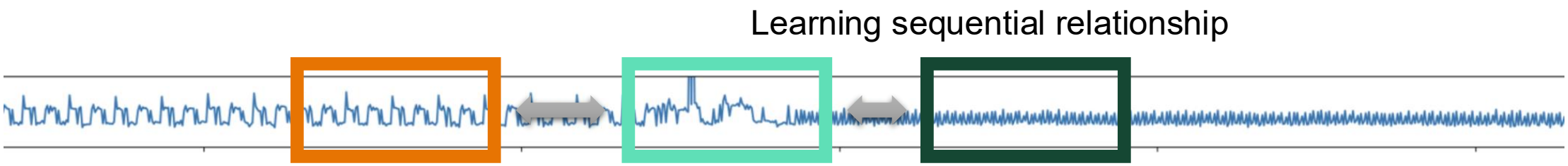


[A segment of latency trace]

Identifying FEF

- **CNN-BiLSTM** extracts patterns and learns sequential dependencies





[A segment of latency trace]

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

Identifying BBEF

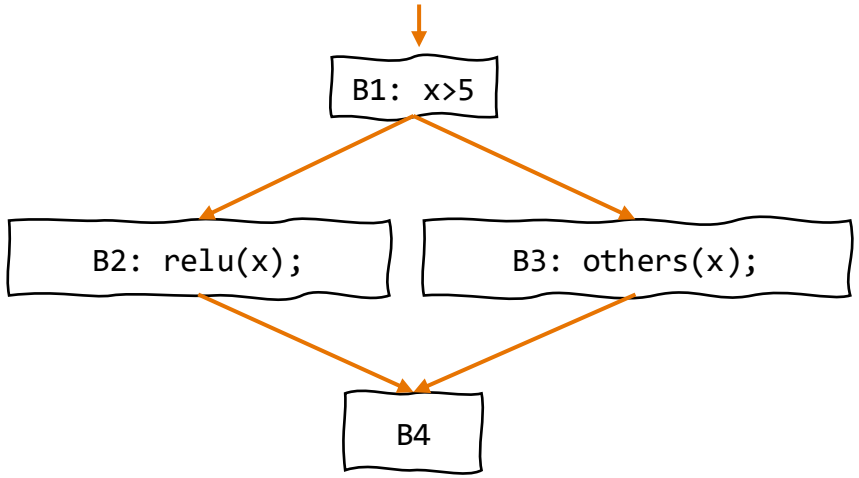
- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

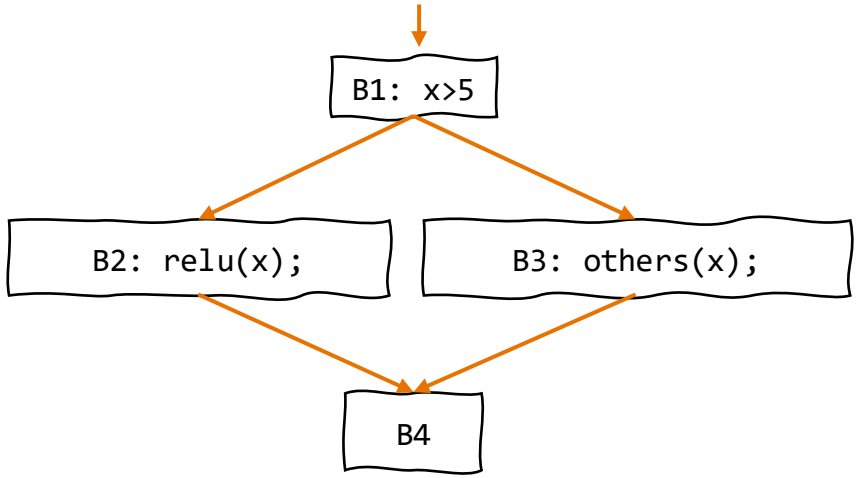
```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```



Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

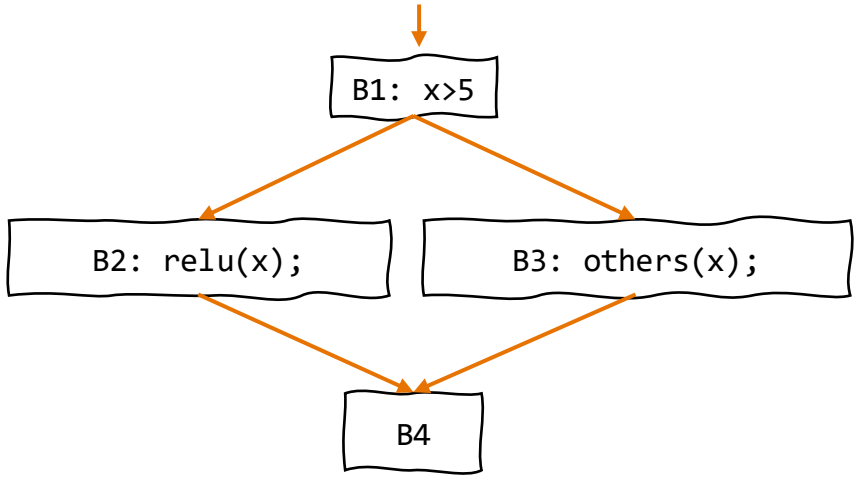
```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```

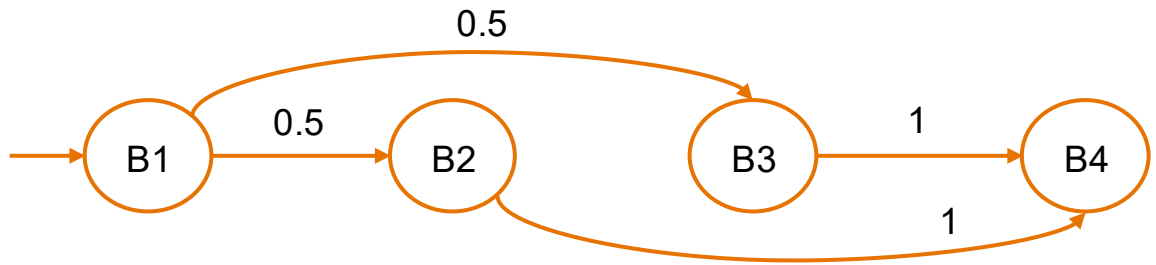


Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```





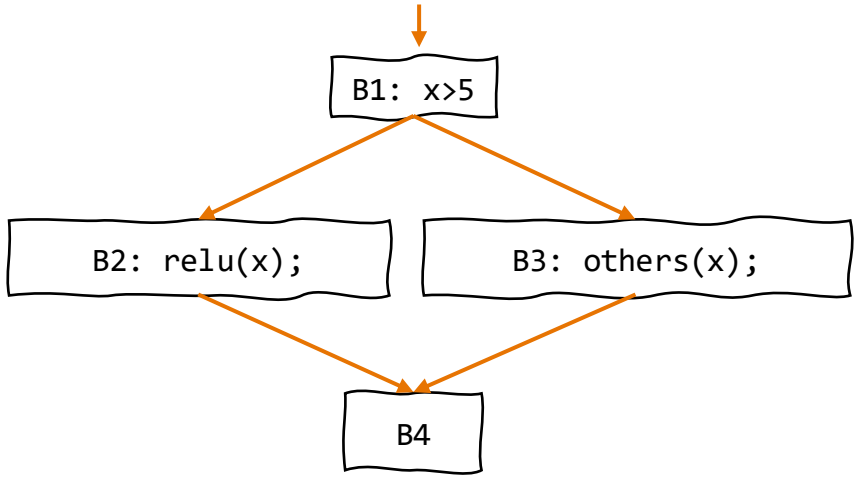
Markov Model

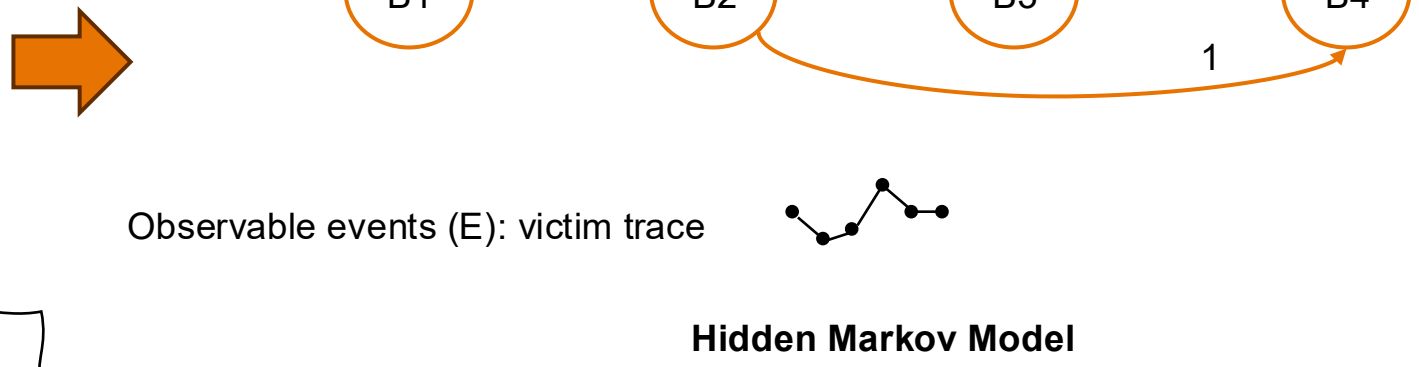
*Markov process? a process whose future state depends **only on** its present state*

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```



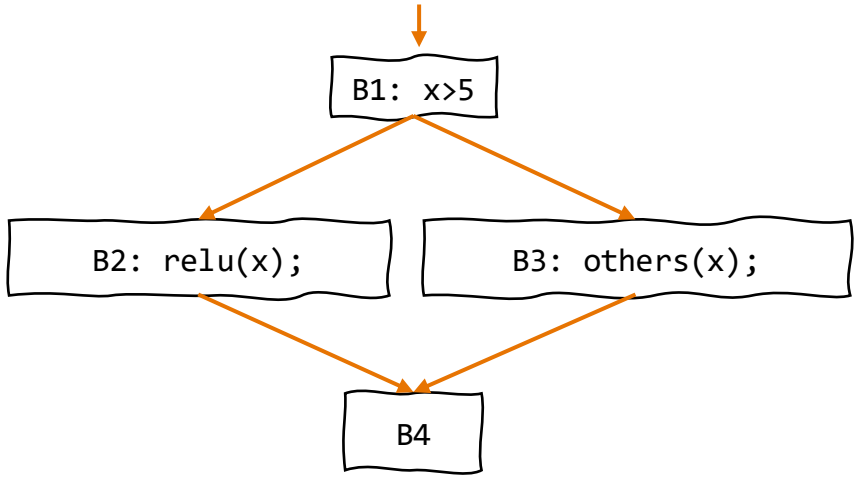


Hidden Markov Model

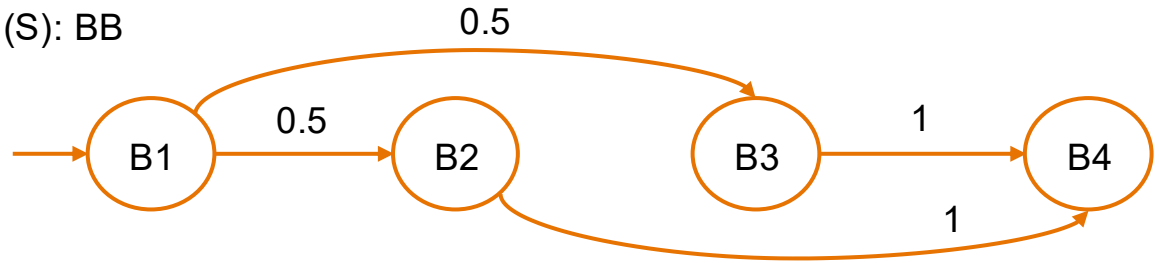
Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```



Hidden states (S): BB



Observable events (E): victim trace

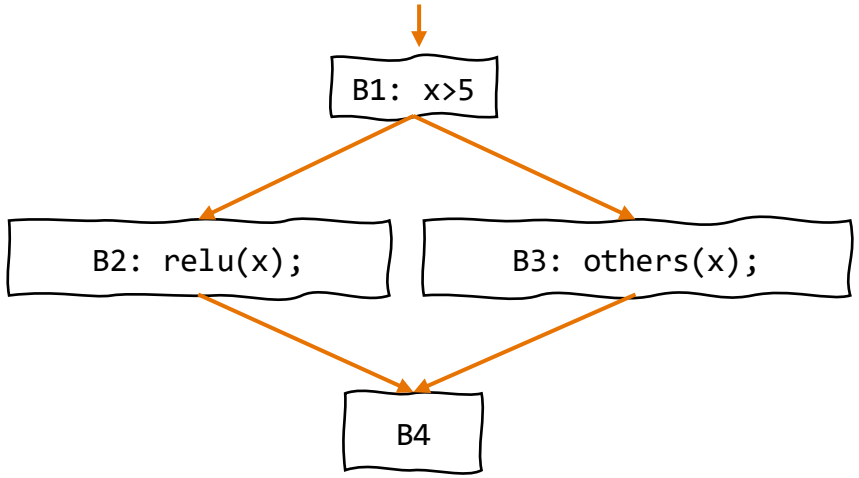
Hidden Markov Model

Emission probability: $P(E_x|S_i)$

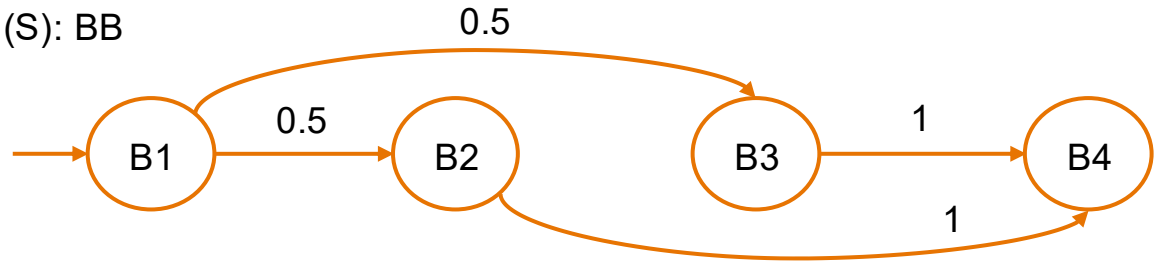
Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

```
{
  ...
  if (is_relu)
    relu(x);
  else
    others(x);
  ...
}
```



Hidden states (S): BB



Observable events (E): victim trace

Hidden Markov Model

Emission probability: $P(E_x|S_i)$ = Normalized Similarity

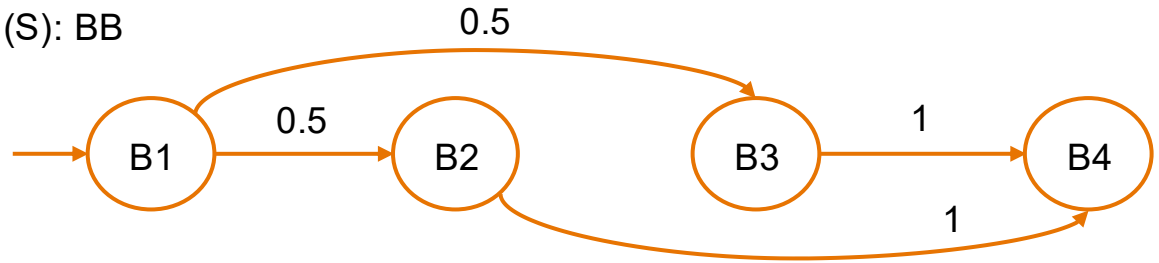
Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

Reference Latency of B2

Reference Latency of B3

Hidden states (S): BB



Observable events (E): victim trace

Hidden Markov Model

Emission probability: $P(E_x|S_i)$ = Normalized Similarity

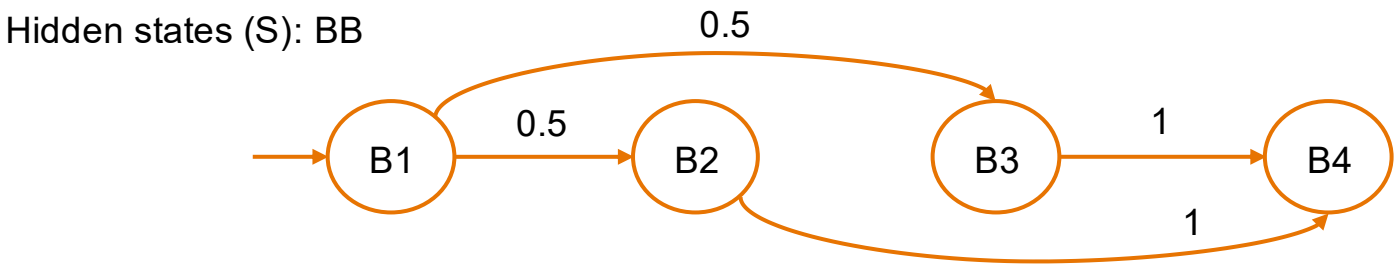
Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

Victim Latency

Reference Latency of B2

Reference Latency of B3



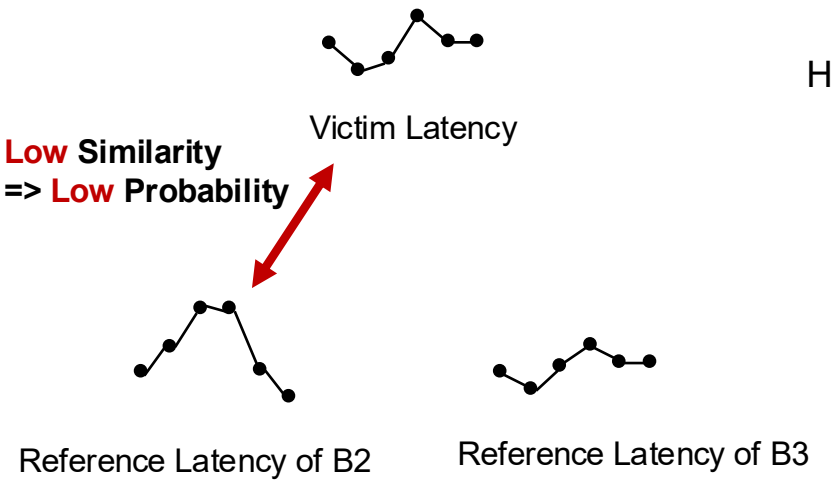
Observable events (E): victim trace

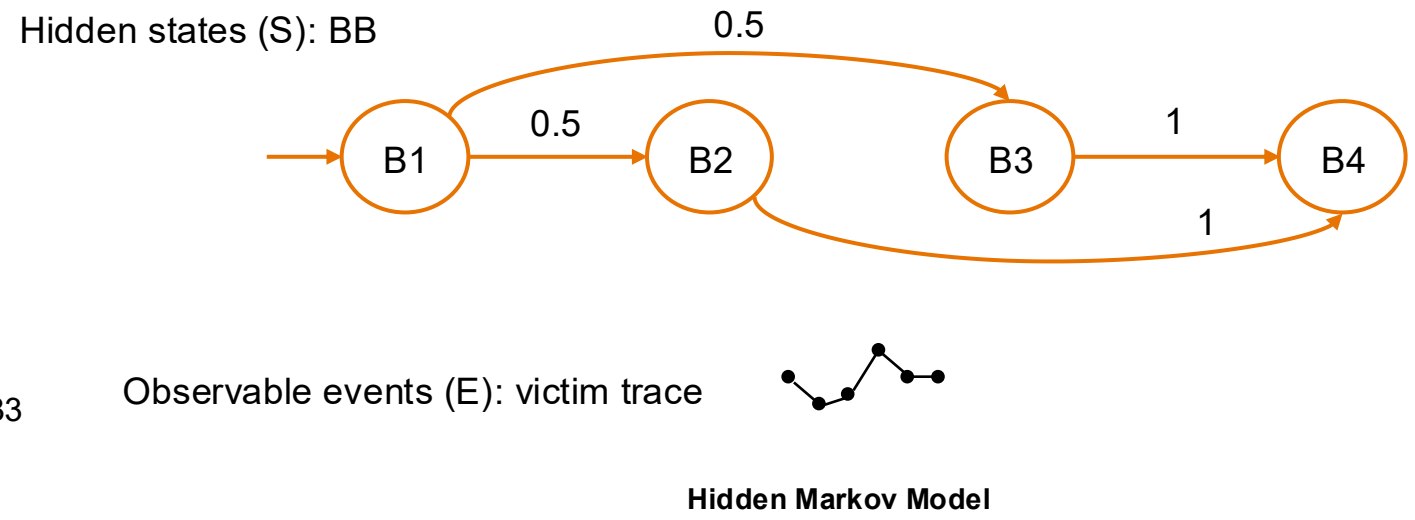
Hidden Markov Model

Emission probability: $P(E_x|S_i)$ = Normalized Similarity

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

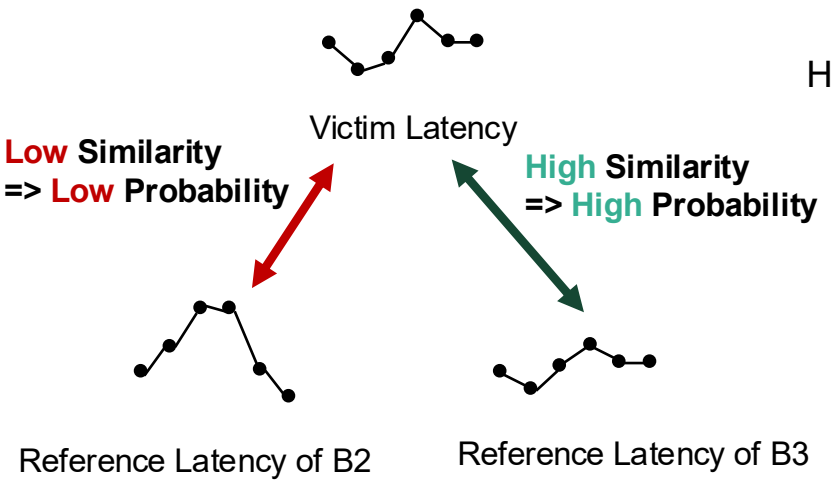


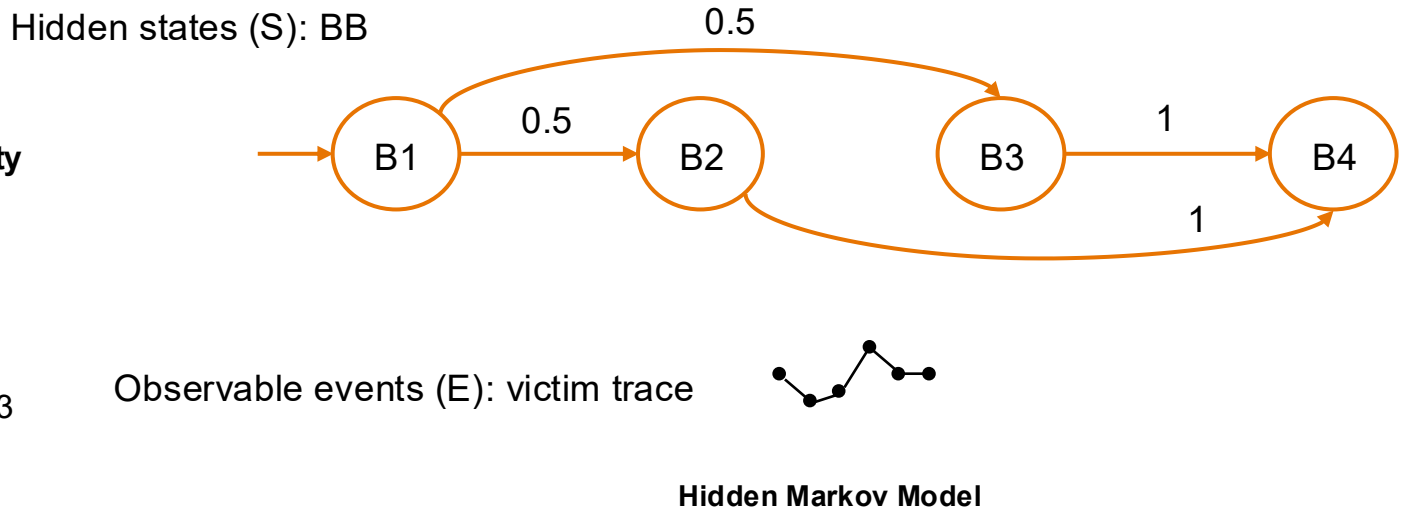


Emission probability: $P(E_x|S_i)$ = Normalized Similarity

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model

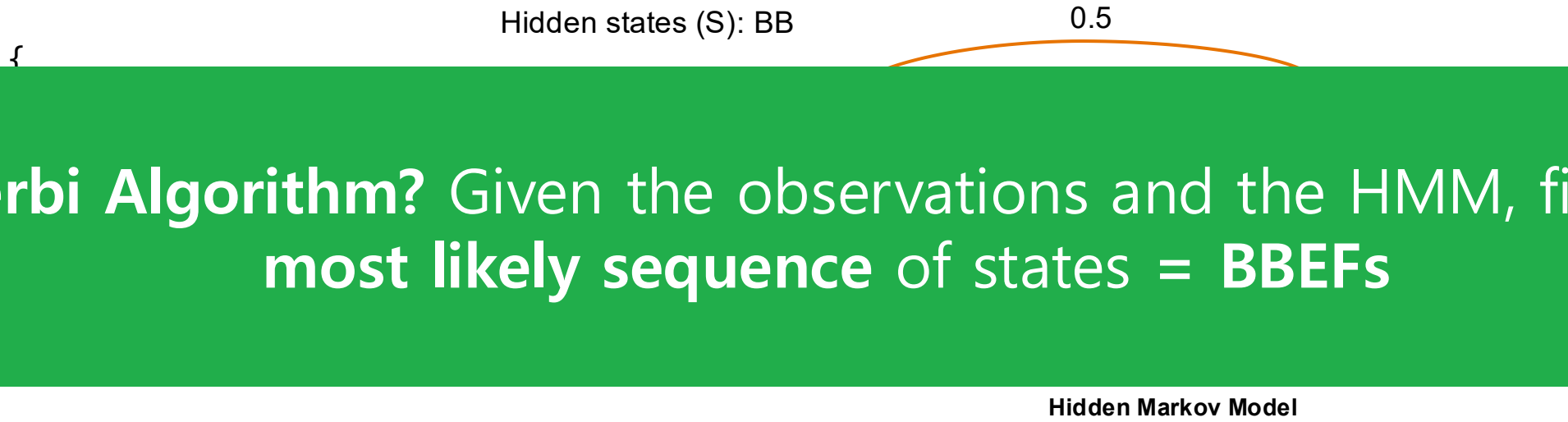




Emission probability: $P(E_x|S_i)$ = Normalized Similarity

Identifying BBEF

- Semi Hidden Markov Model (SHMM) represents control flows as a probabilistic model



Viterbi Algorithm? Given the observations and the HMM, find **the most likely sequence** of states = **BBEFs**

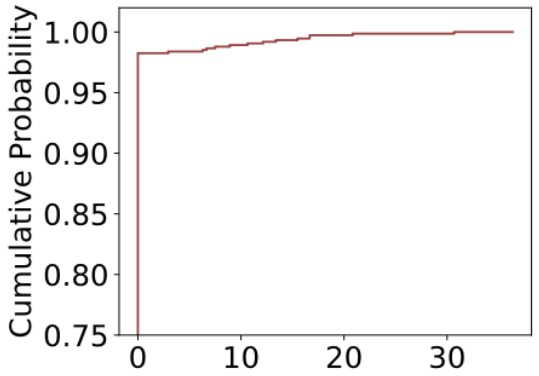
Hidden Markov Model

Emission probability: $P(E_x|S_i)$ = Normalized Similarity

Evaluation: Overview

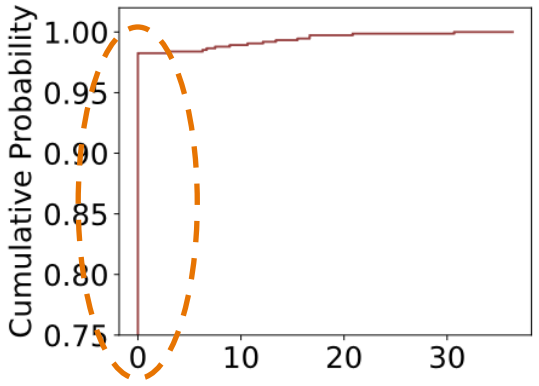
- Setup
  - Intel Core i7-10700 CPU with SGX support
- 3 libraries/frameworks
  - **Darknet**, Tensorflow Lite, ONNX runtime
- 100 random DNN models are tested
  - 24 predefined layer combinations from 10 layer types
    - Convolutional, FC, activation, maxpool, avgpool, softmax, etc.
  - Hyperparameters are randomly chosen
    - #filters, filter size, activation function type, stride, padding, etc.

Evaluation result: Accuracy



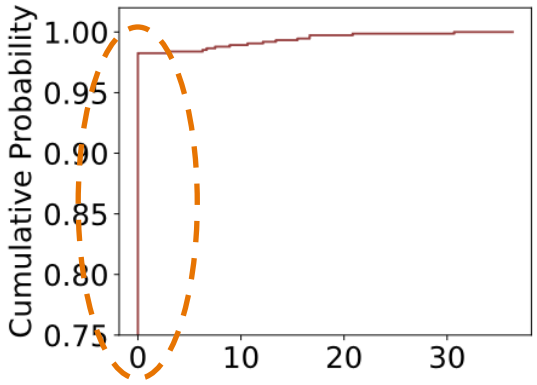
Error rate of Identified FEFs

Evaluation result: Accuracy

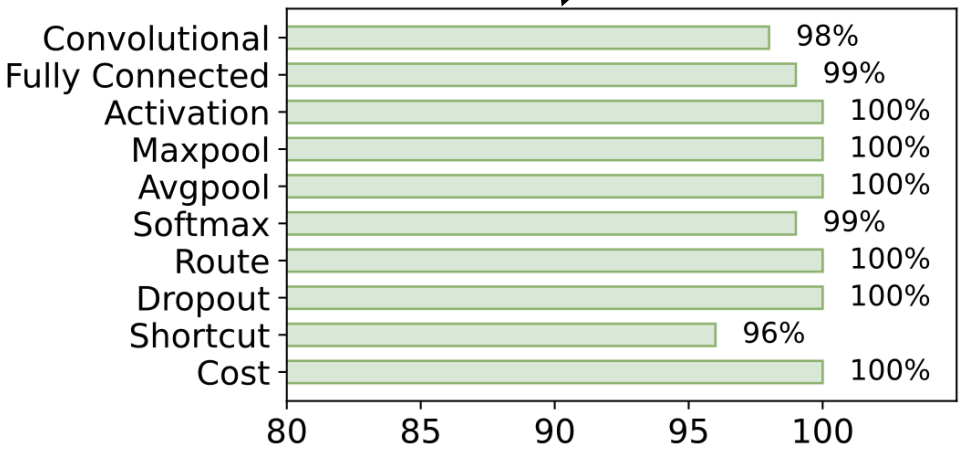


Error rate of Identified FEFs

Evaluation result: Accuracy

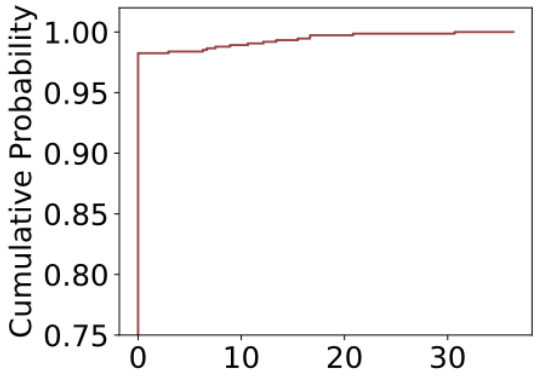


Error rate of Identified FEFs

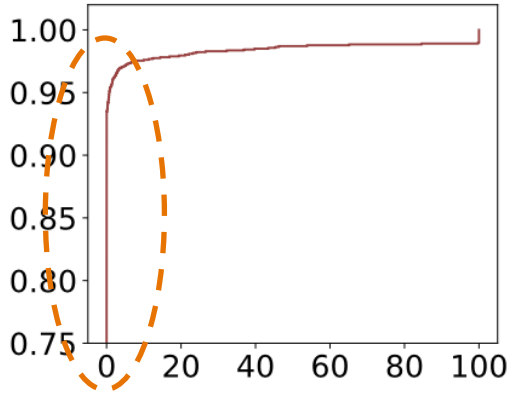


Accuracy of recovered layer types

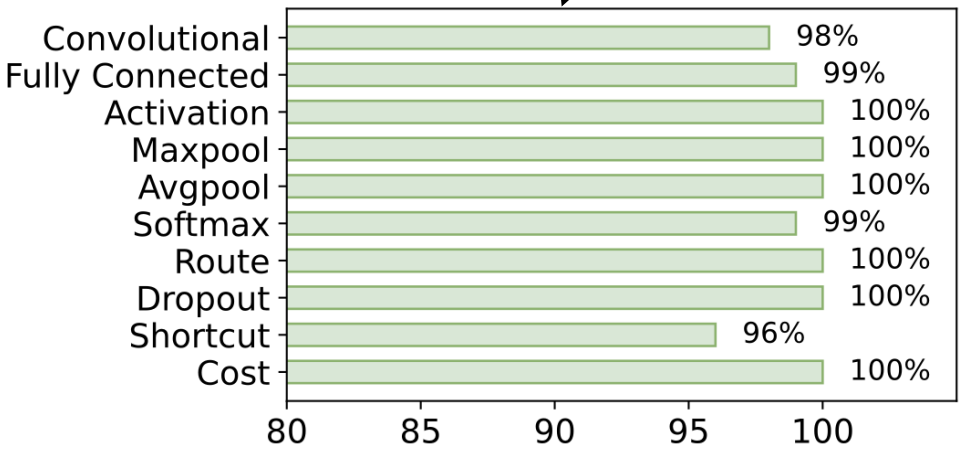
Evaluation result: Accuracy



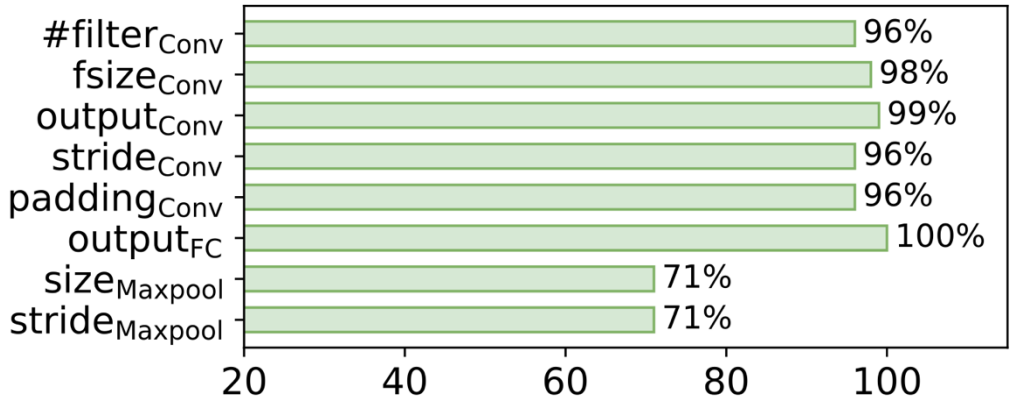
Error rate of Identified FEFs



Error rate of Identified BBEFs

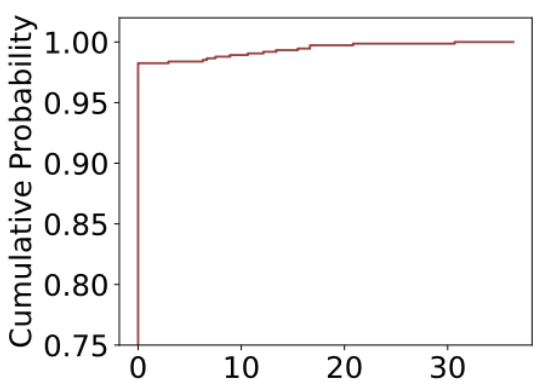


Accuracy of recovered layer types

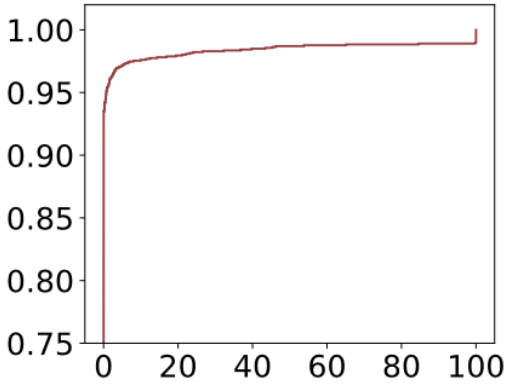


Accuracy of recovered HPs

Evaluation result: Accuracy

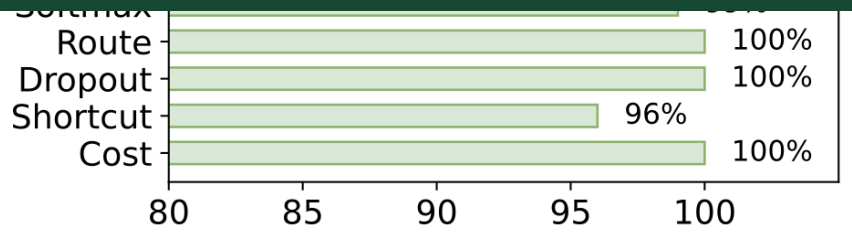


Error rate of Identified FEFs

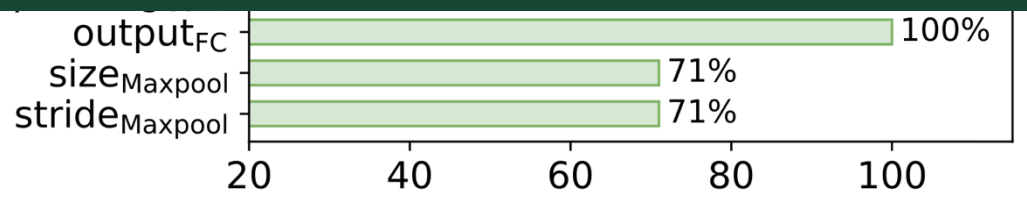


Error rate of Identified BBEFs

Accurate identification of FEFs and BBEFs leads to high accuracy in extracting architecture details



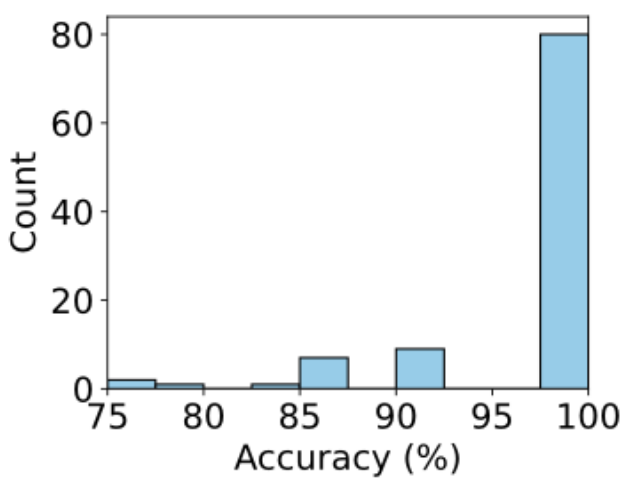
Accuracy of recovered layer types



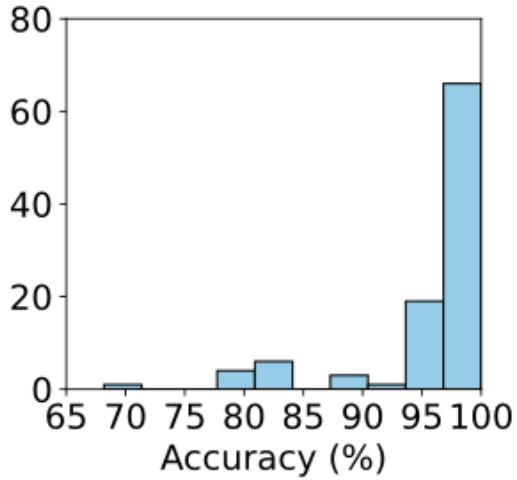
Accuracy of recovered HPs

Evaluation result: Attack accuracy

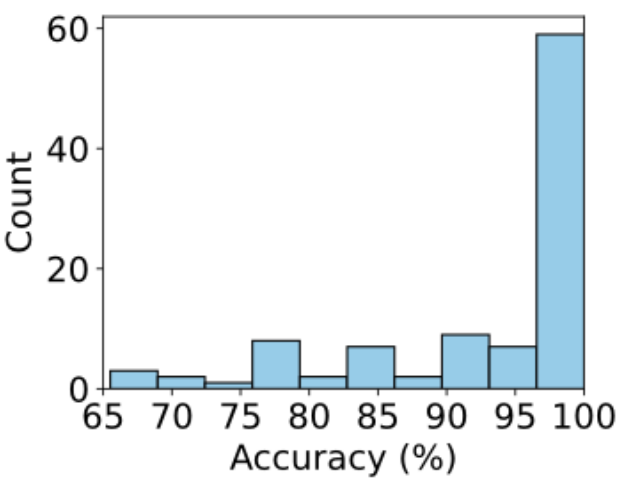
- High attack accuracy
  - Metric = #identified components/#total components
  - Average: 97%, 96%, and 94%, respectively



Darknet



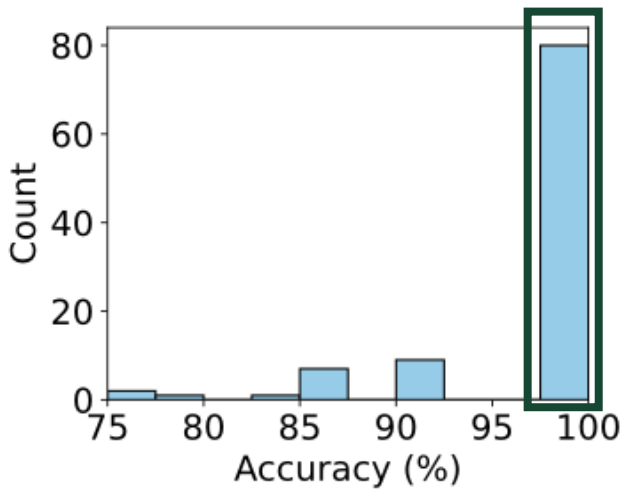
TF Lite



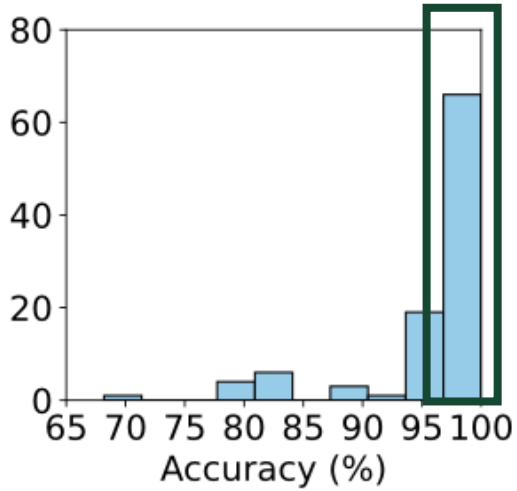
ONNX Runtime

Evaluation result: Attack accuracy

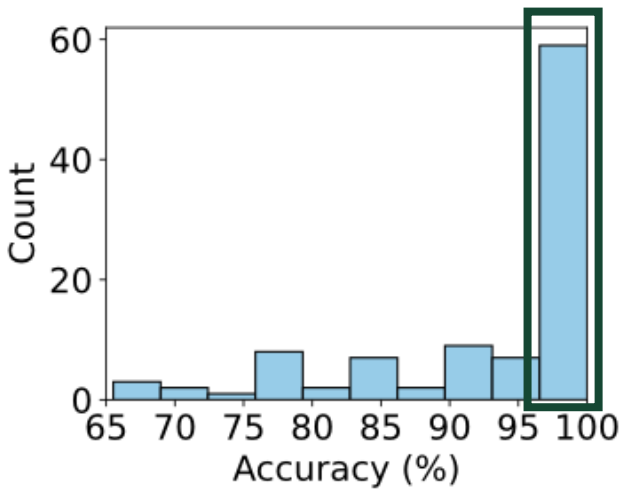
- High attack accuracy
  - Metric = #identified components/#total components
  - Average: 97%, 96%, and 94%, respectively



Darknet



TF Lite



ONNX Runtime

Conclusion

- **DLS** extracts a DNN architecture from latency traces
- To bridge the trace–architecture gap, DLS reconstructs execution flows and maps them to architectural details
  - Function-level execution flows (FEF) via CNN-LSTM
  - Basic block-level execution flows (BBEF) via SHMM
- High attack accuracy (>94%) demonstrates practical effectiveness

Thank you!

Supported by